Digital certificate and PKI

SI Lab

1. Objectives

The learning objective of this lab is to get familiar with the concepts in the Public-Key encryption and Public-Key Infrastructure (PKI). After finishing the lab, students should be able to gain a first-hand experience on public-key encryption, digital signature, public-key certificate, certificate authority, authentication based on PKI.

PKI (Public Key Infrastructure) is a set of physical devices used to create certificates which bind a user public key to its real identity. The concept of the PKI (also known as asymmetric cryptography) is based on the fact that each peer has its own set of private and public key.

2. Lab Tasks:

1. Establish a certificate authority (CA)
2. Issue a digital certificate to a company
3. Set up an HTTPS server using this certificate

3. Lab Environment

Using VirtualBox,we have created a pre-built virtual machine (VM) image for Ubuntu Linux (version 12.04). You can adjust the configuration according to the resources of your host machine (e.g.,you can assign more memory to this VM if your host machine has enough memory):

• Operating system: Ubuntu 12.04

• Memory: 1024M RAM

• Disk space: Maximum 50G disk space

Are two accounts in the VM. The usernames and passwords are listed in the following:

3. User ID: root, Password: power

2. User ID: silabs, Password: security20

Install OpenSSL. OpenSSL is a free and open-source cryptographic library that provides several command-line tools for handling digital certificates. In this lab, we will use openssl commands and libraries. Openssl binaries is already installed in our VM. Install OpenSSL libraries and associated documents by doing the following:

- Download the tar file openssl-1.0.1.tar.gz from the link: www.openssl.org/source/
- Un-tar the tar ball. *Read the INSTALL file* and enter the directory where the package is extracted.

- You need to execute the following commands to install OpenSSL libraries and documents as a root:

```
./config
make
make test
make install
```

1. Become a Certificate Authority (CA)

A certificate authority (CA) is an entity that signs digital certificates. Many websites need to let their customers know that the connection is secure, so they pay an internationally trusted CA (eg, VeriSign, DigiCert) to sign a certificate for their domain.

In some cases it may make more sense to act as your own CA, rather than paying a CA like DigiCert. Common cases include securing an intranet website, or for issuing certificates to clients to allow them to authenticate to a server (eg, Apache, OpenVPN).

In this lab, we need to create digital certificates, but we are not going to pay any commercial CA. We will become a root CA ourselves, and then use this CA to issue certificate for others (e.g. servers). In this task, we will make ourselves a root CA, and generate a certificate for this CA.

While a digital certificate is typically signed by a CA, the certificate of a root CA is self-signed. Root CA's certificates are usually preloaded into most operating systems, web browsers, and other applications that rely on PKI. These certificates are unconditionally trusted. Prepare the directory

Choose a directory (`/root/ca`) to store all keys and certificates. Create a directory `CA` in your home directory

```
# mkdir /root/ca
```

a) Configuration File

In order to use OpenSSL to create certificates, you have to have a configuration file - `openssl.cnf`, located in the following folder `/usr/lib/ssl/openssl.cnf`

- Copy the default configuration file to your directory
- Edit the default `openssl.cnf` file to fit your own setup. You can keep the default settings in most of the sections. However, you need to make changes in the `[ca]` section.

```
[ CA_default ]

dir             = /root/ca              # Where everything is kept
certs           = $dir/certs            # Where the issued certs are kept
crl_dir         = $dir/crl              # Where the issued crl are kept
database        = $dir/index.txt        # database index file.
#unique_subject = no                    # Set to 'no' to allow creation of
                                        # several ctificates with same subject.
new_certs_dir   = $dir/newcerts         # default place for new certs.

certificate     = $dir/cacert.pem       # The CA certificate
serial          = $dir/serial           # The current serial number
crlnumber       = $dir/crlnumber        # the current crl number
                                        # must be commented out to leave a V1 CRL
crl             = $dir/crl.pem          # The current CRL
private_key     = $dir/private/cakey.pem# The private key
RANDFILE        = $dir/private/.rand    # private random number file

x509_extensions = usr_cert              # The extentions to add to the cert
```

Create the directory structure.

```
# cd /root/ca
```

For the index.txt file, simply create an empty file. For the serial file, put a single number in string format (e.g. 1000) in the file. Once you have set up the configuration file openssl.cnf, you can create and issue certificates.

```
# mkdir certs crl newcerts private
# chmod 700 private
```

b) Certificate Authority (CA)

We need to generate a self-signed certificate for our CA. This means that this CA is totally trusted, and its certificate will serve as the root certificate. Run the following command to generate the self-signed certificate for the CA:

```
# openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf
```

You will be prompted for information and a password. The output of the command are stored in two files: `ca.key` and `ca.crt` . The file `ca.key` contains the CA's private key, while `ca.crt` contains the public-key certificate.

```
Country Name (2 letter code) [AU]:RO
State or Province Name (full name) [Some-State]:Romania
Locality Name (eg, city) []:Bucharest
Organization Name (eg, company) [Internet Widgits Pty Ltd]:ATM
Organizational Unit Name (eg, section) []:ATM Certificate Authority
Common Name (e.g. server FQDN or YOUR name) []:ATM Root CA
Email Address []:admin@atm.ca
```

c) Verify the root certificate

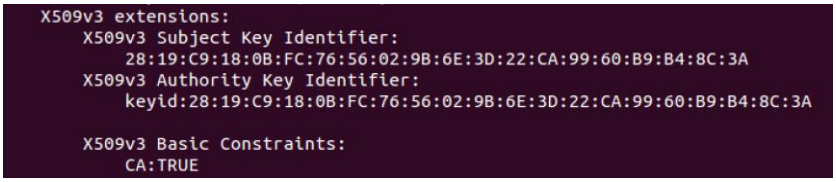
```
# openssl x509 -noout -text -in ca.crt
```

The output shows:

- the `Signature Algorithm` used
- the dates of certificate `Validity`
- the `Public-Key` bit length
- the `Issuer`, which is the entity that signed the certificate
- the `Subject`, which refers to the certificate itself

The `Issuer` and `Subject` are identical as the certificate is self-signed. Note that all root certificates are self-signed.

The output also shows the **X509v3 extensions**. We applied the `v3_ca` extension, so the options from `[v3_ca]` should be reflected in the output.



2. Create a Certificate for `PKISILab.com`

Now, we become a root CA, we are ready to sign digital certificates for our customers. Our first customer is a company called `PKISILab.com`.

a) Generate public/private key pair

```
# openssl genrsa -aes128 -out server.key 1024
```

Take a look at the content of the certificate with the following command:

```
# openssl rsa -in server.key -text
```

b) Generate a Certificate Signing Request (CSR)

The next step is to create the CSR. But make sure that you enter the domain name of the server as Common Name:

```
# openssl req -new -key server.key -out server.csr -config openssl.cnf
```

```
Country Name (2 letter code) [AU]:RO
State or Province Name (full name) [Some-State]:Romania
Locality Name (eg, city) []:Bucharest
Organization Name (eg, company) [Internet Widgits Pty Ltd]:ATM
Organizational Unit Name (eg, section) []:ATM_PKI
Common Name (e.g. server FQDN or YOUR name) []:PKISILAB.com
Email Address []:admin@pki
```

c) Generating Certificates

The CSR can now be signed by the Root CA:

```
# openssl ca -in server.csr -out server.crt  -cert ca.crt -keyfile ca.key -
config openssl.cnf
```

Explanation of the command:

```
ca                      certificate authority management
-in <file>              input file (the CSR)
-out <file>             output file (signed certificate)
-config <file>          use the given openssl config file
```

d) Verify the certificate

```
# openssl x509 -noout -text -in server.crt
```

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4096 (0x1000)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=RO, ST=Romania, L=Bucharest, O=ATM, OU=ATM Certificate Authority, CN=ATM Root CA/emailAddress=admin@atm.ca
        Validity
            Not Before: Oct  8 14:09:51 2017 GMT
            Not After : Oct  8 14:09:51 2018 GMT
        Subject: C=RO, ST=Romania, O=ATM, OU=ATM_PKI, CN=PKISILAB.com/emailAddress=admin@pki
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (1024 bit)
```

3. Using the PKI and Certificate for Securing a Web Site

In this section, we will explore how the PKI and certificates can be used by a web server to secure web browsing. To make it simpler, please add the following entry to the /etc/hosts file on the Linux computer:

```
127.0.0.1 PKISILAB.com
```

This will allow an application to resolve the IP address of the `PKISILAB.com` domain. Next, we need to launch a simple web server with the certificate generated in the previous section. Please do the following:

a) Combine the secret key and certificate into one file:

```
# cp server.key server.pem
```

```
# cat server.crt >> server.pem
```

b) Launch a simple web server (HTTPS server) with the server.pem certificate by using the OpenSSL s_server command:

```
# openssl s_server -cert server.pem -www
```

TO DO:

1. Revoke a certificate

   The **openssl ca -revoke** command marks a certificate as revoked in the CA database

2. Create a certificate revocation list
   The **openssl ca -gencrl** command creates a certificate revocation list (CRL).